PUBLIC REPORT

1inch Protocol (formerly 1split) Audit

peppersec

Foreword

Status of a vulnerability or security issue

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

No issue

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the label.

✓ Fixed

The fixed security issues get the label that informs those reading our public report that the flaws in question should no longer be worried about.

Addressed

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice tag right in front of it.

Acknowledged

If an issue is planned to be addressed in the future, it gets the tag, and a client clearly sees what is yet to be done.

Although the issues marked "Fixed" and "Acknowledged" are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

Severity levels

We also rank the magnitude of the risk a vulnerability or security issue pose. For this purpose, we use 4 "severity levels" namely:

1. Minor
2. Medium
3. Major
4. Critical

More details about the ranking system as well as the description of the severity levels can be found in Appendix 1. Terminology.

TABLE OF CONTENTS

01. Introduction

SCOPE OF WORK

The auditors were provided with a single file OneSplitAudit.sol. The rest of the repo was out of the scope of the audit.

The team put forward the following assumptions regarding the security of the OneSplitAudit contract:

- It is safe to have infinite approves of any tokens to this smart contract since it can only call transferFrom() with the first argument equal to msg.sender.
- It is safe to call swap() with a reliable minReturn argument. If the returning amount does not reach the value of minReturn, the whole swap will be reverted.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

SECURITY ASSESSMENT METHODOLOGY

The smart contract's code is scanned both manually and automatically for known vulnerabilities and logic errors that can lead to potential security threats. The conformity of requirements (e.g., White Paper) and practical implementations are reviewed as well on a consistent basis. More about the methodology can be found here.

AUDITORS

1. PepperSec

02. Discovered issues

CRITICAL ISSUES

During the audit, PepperSec's experts found **no Critical issues** in the code of the smart contract.

MAJOR ISSUES

During the audit, PepperSec's experts found **no Major issues** in the code of the smart contract.

MEDIUM ISSUES

1 `msg.value` validation

Severity: **MEDIUM**

The swap method does not check `msg.value` for any transfers (ETH to a token, token to token).

Recommendation:

1. Consider adding explicit `require` that handles `msg.value`.

Status:

✓ Fixed Link

2 Token balances validation

Severity: **MEDIUM**

The swap method expects `fromToken` and `toToken` balances to be zero, which negatively affects the security of the smart contract.

Recommendation:

1. Consider wrapping the `oneSplitImpl.swap` by the `balanceBefore` and `balanceAfter` checks. Thus, `returnAmount` could be calculated as `balanceAfter - balanceBefore`.

Status:

✓ Fixed with commits 1 and 2.

3 Claim stuck and spam tokens

Severity: **MEDIUM**

It is assumed the contract does not have any tokens before and after a swap. In any case, it is useful to have a special method that allows retrieving stuck tokens.

Recommendation:

1. Consider adding the corresponding special method. Make sure the method does not use any approves.

Status:

✓ Fixed Link

4 `fromToken` is not equal to `toToken` check

Severity: **MEDIUM**

The logic underlying the `swap` method checks that `fromToken != toToken`. However, the `swap` method itself is a better place for that kind of require. This type of check is common for any underlying implementation.

Recommendation:

1. Consider moving the check to the `swap` method.

Status:

✓ Fixed Link

MINOR ISSUES

1 Redundant goodSwap method

Severity: **MINOR**

The goodSwap method is a combination of the getExpectedReturn and swap methods. Since getExpectedReturn is supposed to be called from the DApp frontend rather than from a smart contract, it can be confusing for developers who utilize OneSplit.

Recommendation:

1. Consider removing the goodSwap method and writing proper documentation for the methods.

Status:

✓ Fixed Link

Conclusion

The main goal of the audit was to verify the claims regarding the security of the smart contract (see the Scope of work section). According to the code, it reverts if the `minReturn` amount of buying token is not met. That said, the OneSplit contract does not contain arbitrary logic that could be exploited to transfer another user's token. The contract calls the `transferFrom` method with `msg.sender` as from address and with the exact `amount` the user provides. Both claims appear valid.
During the audit, the experts discovered several Medium and Minor issues that were later fixed by the team. There were no Critical or Major issues found.

Appendix 1. Terminology

1 Severity

Assessment the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood of exploitation		

MINOR

Minor issues are generally subjective in nature or potentially associated with topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

MEDIUM

Medium issues are generally objective in nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.

MAJOR

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

CRITICAL

Critical issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems or, ultimately, a full failure in the operations of the contract.

About Us

Worried about the security of your project? You’re on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you’ve just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We’re ready to battle for the security of your project.

LET’S KEEP IN TOUCH

peppersec.com

hello@peppersec.com

Github